KATHOLIEKE UNIVERSITEIT LEUVEN

Faculteit der Economische en Toegepaste Economische Wetenschappen

The conversion of limited-entry decision tables to optimal and near-optimal flowcharts : two new algorithms

M.VERHELST

Onderzoeksrapport Nr. 7204

This report is scheduled to appear in Communications of the ACM.

The conversion of limited-entry decision tables to optimal and near-optimal flowcharts: two new algorithms

M. Verhelst (*)

Abstract

In this paper, two new algorithms for deriving optimal and near-optimal flowcharts from limited-entry decision tables are presented. Both take into account rule frequencies and the time needed to test conditions. One of the algorithms - called the optimum-finding algorithm - leads to a flowchart which truly minimizes execution time for a decision table in which simple rules are already contracted to complex rules. The other one - called the optimum-approaching algorithm - requires much less calculations but does not necessarily produce the optimum flowchart. The algorithms are first derived for treating decision tables not containing an ELSE-rule, but the optimum-approaching algorithm is shown to be equally valid for tables including such a rule.

Both algorithms are compared with already existing ones and are applied to a somewhat large decision table derived from a real case. From this comparison two conclusions can be drawn. 1. The optimum-approaching algorithm will usually lead to better results than comparable existing ones and will not require more - but usually less - computation time. 2. In general, the greater computation effort needed for applying the optimum-finding algorithm will not be justified by the small reduction in execution time obtained.

Key words and phrases: decision table, flowcharting, preprocessor, optimal programs, search

CR categories: 3.50, 3.59, 4.19, 4.29, 4.49, 5.31

(*) M. Verhelst is assistant professor at the Institute of Applied Economic Sciences, Katholieke Universiteit te Leuven, Belgium.

1. Introduction

In the past, a number of algorithms have been proposed for converting decision tables to computer programs.

From the vieuwpoint of the techniques used for the conversion, they can be divided into two categories: whose deriving a flowchart, (viz. the algorithms proposed by Pollack [5], Reinwald and Soland [6] and Shwayder [7] and those using rule mask techniques (viz. the algorithms proposed by King [2], Kirk [3], Muthukrishnai and Rajaraman [4] and Verhelst [8].

From the vieuwpoint of the result obtained, the algorithms can be classified into three categories: those deriving optimal solutions in terms of execution time of the resulting program or in terms of storage space occupied by that program [6], [8], those deriving near-optimal solutions [2], [5], [7] and those not aiming at optimization [3], [4].

In the present paper, two new algorithms will be proposed. They both are using the flowcharting method. One of them is leading to an optimal solution and the other one to a near-optimal solution, in terms of execution time of the resulting program. In what follows, it is indeed argued, that optimization of storage space is not relevant.

After having presented the algorithms, we shall compare them to the algorithms of Pollack and Shwayder in terms of their computational efficiency and their result. The comparison with the Reinwald and Soland algorithm is not carried through, because the input of the latter algorithm is different from the input of all others. Indeed, the input to the Reinwald and Soland algorithm is a complete decision table containing only simple rules (i.e. rules not showing dashes), whereas the input of the others is a decision table which contains dashes. For that reason, the concept of optimization is somewhat different.

In understanding the comparison, the reader is supposed to be familiar with the algorithms of Pollack [5] and Shwayder [7]. A treatment of the fundamentals of the construction and use of decision tables can be found in [1].

2. Minimizing criteria

The algorithms for translating decision tables into optimal flow-charts are utilizing one of two criteria: minimization of storage space or minimization of execution time. The algorithms described in this paper will only use as a criterion minimization of execution time. Restriction to this criterion is based on the following practical consideration.

If the conditions specified in a decision table are such that only one or two machine language instructions are needed to state each of them, minimization of storage space is not so important. If at the other extreme, each condition is quite voluminous and a subroutine is needed for testing it, the program can be structured such that each subroutine only appears at one storage space. Testing the condition at a particular point of the flowchart then means to branch to the subroutine and to return to the appropriate point. In the latter case, testing a condition can be specified by a couple of machine language instructions, which again means that the minimization problem is no longer important.

For a clear understanding of the rationale behind the algorithms proposed in this paper, it is necessary to introduce and consider a number of concepts. For this reason, the next three sections will treat the following subjects: the use of dashes and stars in a decision table, the lower bound for the test time of a decision table and finally the basic objective of the algorithms.

3. The use of dashes and stars in a decision table

The condition part of the rules in a (limited-entry) decision table usually shows the following symbols: Y(yes), N(no) and - (don't care). The dash means that the question is irrelevant in the context of the considered decision rule. It is very important to distinguish between two reasons for the irrelevance of a particular question. A first reason may be that the answer can be Y or N, but that it does not matter; a second reason may be that the answer to the question is known because it is implied by the answers to the other questions explicitly stated in the same rule. In the latter case, we propose not to use the dash, but to indicate the real answer surmounted by a star. The following example illustrates this way of notation.

	R_1	R_2
Age < 20	Y	N*
Age > 60	N*	Y

The reason for using a star in this case is related to the fact, that the use of a dash sometimes leads to pairs of dependent rules, which cannot be handled by algorithms for converting decision tables to flowcharts.

4. The lower bound for the test time of a decision table

In order to calculate the test time of a decision table, we need information about the relative frequencies of the rules and the test time of the conditions. If the decision table does not contain an ELSE-rule, it can be shown that a flowchart derived from the decision table can never require a test time which is lower than the time obtained by testing all answers not dashed and not starred. This lower bound will be denoted by S. If the frequency for rule j is denoted

by f_j (j = 1, ...,n) and the test time of condition i by t_i (i = 1, ...,m), then S can be calculated as follows:

$$S = \sum_{i=1}^{m} \left[t_i \sum_{j} \right] f_j$$, for all j for which the answer on condition i is not dashed or starred.

As an example, consider the calculation of S for the following decision table

	R_1	R_2	R_3	
f_j	0.20	0.50	0.30	t_i
C_1	N	N	Y	20
C_2	N	Y	Y*	10
C_3	N*	-	Y	5

S = 20 (0.20 + 0.50 + 0.30) + 10 (0.20 + 0.50) + 5 (0.30) = 28.5. This simply means that the average time to test a transaction can never be less than 28.5, whatever the structure of the flowchart be.

In order to prove this theorem, let us consider the way in which the flowcharting method operates. The action part of a particular rule is reached when the algorithm produces an empty subtable or a subtable consisting of 1 column. Theoretically, it would be possible that not all conditions in this column are dashed or starred. Assuming that the table does not contain an ELSE-rule, this would then mean that for that rule at least one condition which is not starred or dashed does not have to be tested, in which case S would be lower than stated above. If however such a case would arise, it would mean that in preparing the table, one has forgotten to star those conditions. Indeed, if N or Y elements appear in a subtable of one rule, and if the decision table contains no ELSE-rule,

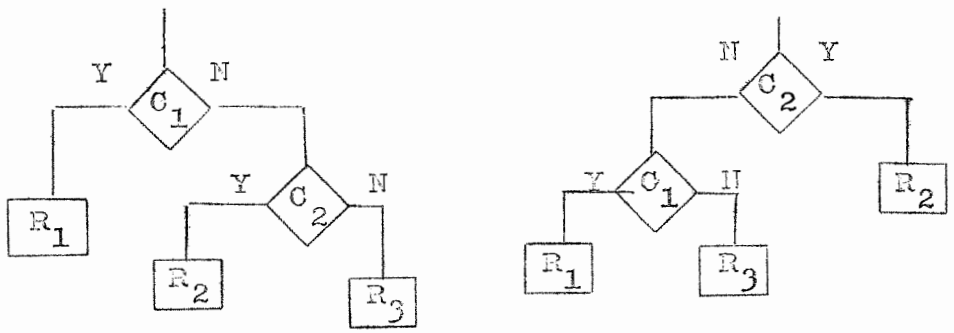
this indicates that, given the combination of answers on conditions tested leading to the subtable of one column, no other rule existe for which the answer on the not yet tested conditions is different. This automatically means that the considered N and Y elements could have been starred in the original table. When such a case arises, one shouhd star these answers and repeat the algorithm.

By proving that the conditions have to be tested for all rules in which they are not dashed or starred, we have delivered the proof that no flowchart can give us a lower average test time than S. This does not mean however that there exists a possible flowchart structure reaching this minimum. Consider as an example the following decision table.

	R_1	R_2	R_3	
f_j	0.50	0.20	0.30	t_i
C_1	Y	N*	N	1
C_2	N*	Y	N	1

$S = 1(0.80) + (0.50) = 1.30$

There are only 2 possible flowchart structures :



Test time : 1 (1.00) + 1 (0.50) = 1.50
Test time: 1 (1.00) + 1 (0.80) = 1.80

5. Basic objective of the algorithms

The algorithms presented in this paper aim at deriving a flowchart which realizes as close as possible the lower bound S. Two algorithms will be proposed. The first one cannot be said to lead to the minimum realizable test time, but will approach this minimum. The second one however minimizes the test time, but requires more search time. The first algorithm, called hereafter "the optimum-approaching algorithm" can be compared with the ones proposed by Pollack [5] and Shwayder [7] ; it also considers only one subtable for optimization purposes, but will be shown to be superior to the ones just mentioned in that it allows the derivation of flow charts of tables containing dependent rules and that it takes into consideration the test times of the conditions. The second algorithm, called hereafter the "optimum-finding" algorithm can be compared to the one proposed by Reinwald and Soland [6] . We think that it can be said to be superior to the Reinwald algorithm, because it requires less search time.

6. The optimum-approaching algorithm

As the reader will recall, any method for converting decision tables into flowcharts starts by selecting a certain condition to be tested and produces from the original decision table two subtables, each containing one row less than the original one and containing the rules for which the answer on the tested condition was not yes (first subtable) or not no (second subtable).

If the purpose is to minimize test time, we should avoid to test in the first place conditions which are not relevant in a high percentage of cases, because the longer we wait to

test such conditions, the more chance we have not to have to test them at all. Also, when two conditions A and B are relevant in an equal number of cases, but condition B takes more time to test, we will prefer to test A and to postpone the testing of B.

Therefore, an algorithm which would approach the lower bound for the test time, can be stated as follows.

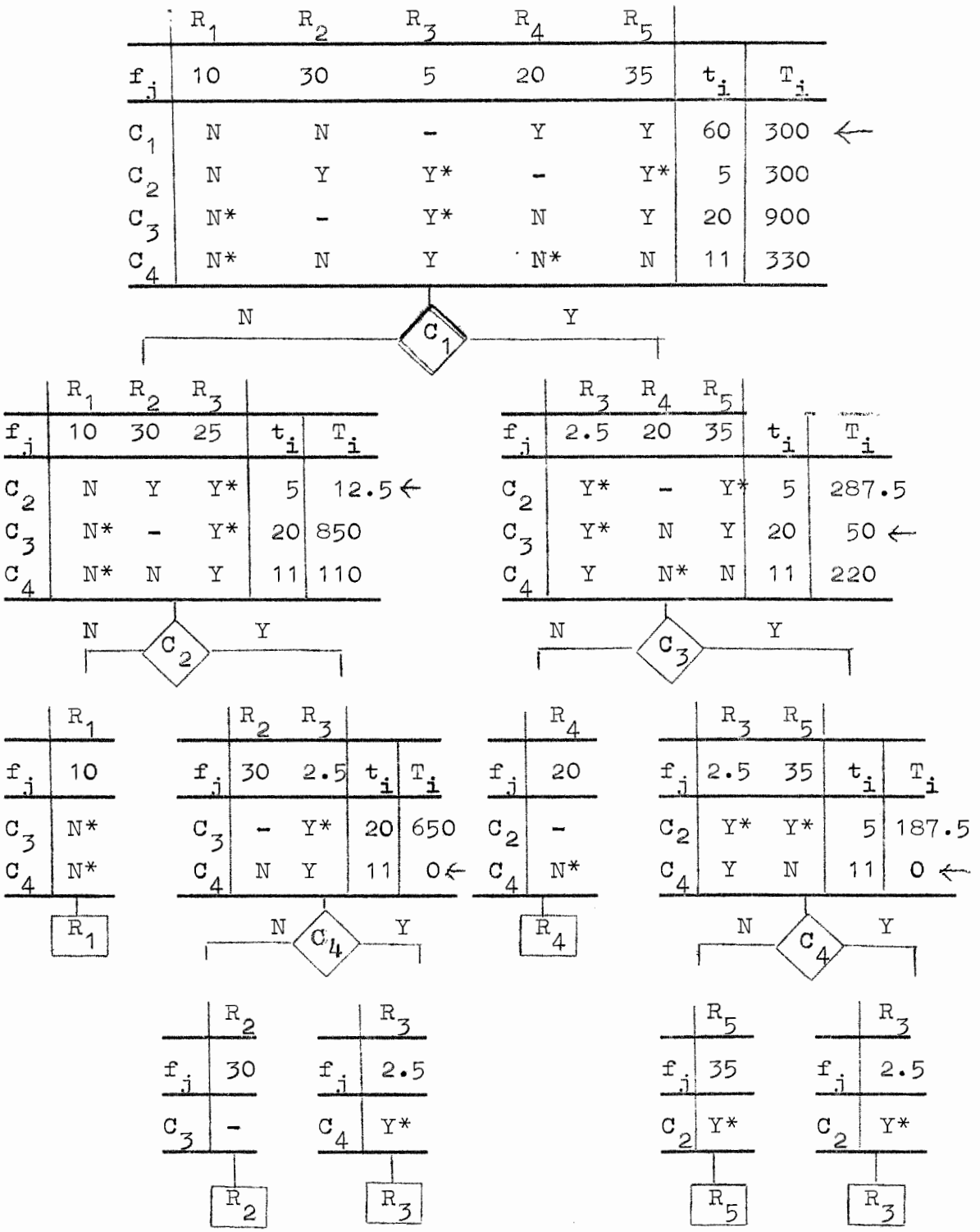
- At each selection stage, compute for each condition row

$$T_i = t_i \sum_j f_j$$ for all j containing a star or a dash in the considered row.

- Select as the condition to be tested at that selection stage the one stated in the row with minimal T_i. If there is a tie, select one of the tied rows. If the selected condition is dashed for one or more rules, divide the frequencies of the columns containing a dash for that condition equally over the two resulting subtables.

The algorithm stops when each branch of the flowchart has empty subtables or subtables with not more than one decision rule.

As an illustration, the algorithm is applied on a decision table as appears in figure 1.



Average test time: S + 300 + 12.5 + 50 + 0 + 0 = S + 362.5

Figure 1

At this point, it might be instructive to compare the results obtained by applying this algorithm to the ones obtained by means of the algorithms of Pollack and Shwayder. Since the latter algorithms do not consider differences in condition test times (t_i), we computed for both cases the flowchart for the above decision table, where the values of t_i were all set equal to 1. The average test time for a transaction then was 2.9 units for Pollack's method and 2.7 units for Shwayder's method and ours. Also, the latter two methods produced the same flowchart. From this, it cannot be concluded however that Shwayder's algorithm and ours would always produce exactly the same results.

7. Theorems on which the optimum finding algorithm is based

Before stating the algorithm producing the tree leading to the minimum average test time, it is necessary to consider a number of underlying theorems, which are at the same time definitions.

Theorem 1 : The minimum increase of test time L_k by testing first C_k can be calculated as the sum of 3 quantities T :

- T_k in the original table;
- min T_i in the subtable obtained for the exit N of C_k;
- min T_i in the subtable obtained for the exit Y of C_k.

The proof of this theorem is simple. By testing C_k, S is immediately increased by T_k. Furthermore, since at the next stage after having tested C_k, necessarily not more than 2 conditions have to be tested, the average test time cannot, at this stage, be increased by less than the sum of the separate minimum T_i's appearing on each side of the node at the next stage of the tree.

Referring as an example to the table of Figure 1, L_1 = 300 + 12.5 + 50 = 362.5

Theorem 2: An upper bound for the increase of test time (M_k) by testing first C_k can always be calculated by applying the optimum-approaching algorithm to a tree starting with C_k. This is obvious and needs no proof.

As an example, for the table of Figure 1, $M_1 = 362.5$

Theorem 3: If at a certain node of a tree under construction, two conditions C_k and C_ℓ are considered as candidates to be tested at that node, and $M_\ell < L_k$, continuing the structure by testing C_k at that point can never lead to an optimal tree.

The proof is again obvious. Since $M_\ell < L_k$, we know that continuing with C_ℓ will always lead to a lower test time than continuing with C_k.

Consider again the table of Figure 1 as an example. At the first node, we see that $L_3 \geqslant 900$; since we know already that $M_1 = 362.5$, we can derive that $M < L_3$. Therefore the optimal tree cannot start with C_3.

Theorem 4: If it is conditionally stated that a tree has to start with C_k at the first node, the tree is optimal if for all conditions ℓ tested at each subsequent node, $M_\ell \leqslant L_i$ for all $i \neq \ell$.

This theorem is nothing else but a corrolary of theorem 3.

Figure 1 can again be referred to for the purpose of illustration. The tree of Figure 1 contains 5 nodes, spread over 3 stages. Let us number these nodes from the left to the right as follows:

- Stage 1 : node 1 ;
- Stage 2 : nodes 2 and 3 ;
- Stage 3 : nodes 4 and 5.

At node 2 : $M_2 = 12.5 + 0 = 12.5$; $L_3 \geqslant 850$; $L_4 \geqslant 110$

At node 3 : $M_3 = 50 + 0 = 50$; $L_2 \geqslant 287.5$; $L_4 \geqslant 220$
At node 4 : $M_4 = 0$; $L_3 \geqslant 650$
At node 5 : $M_4 = 0$; $L_2 \geqslant 187.5$

Therefore the tree of Figure 1 shows the optimal flowchart if this flowchart should start by testing C_1. It should be clearly kept in mind however, that this is not necessarily the overall optimal flowchart, as it may also be possible that the overall optimal flowchart starts with another condition than C_1.

8. The optimum-finding algorithm

On the basis of the theorems considered above, the optimum-finding algorithm can now be stated as follows.

For a decision table containing m conditions, consider a maximum of m stages and proceed as follows for each stage, starting by stage 1.

1. Consider each (sub)table belonging to the stage and fulfulling each of the following conditions:
   - the (sub)table does not belong to a tree already eliminated;
   - it is not tagged for optimality;
   - it does not belong to a tree already tagged for optimality

   For each of these (sub)tables proceed as follows.
   1.1 Compute L_i for each C_i in the (sub)table.
   1.2 Consider k=i for $L_{i_{min}}$.
   1.3 Construct a (sub)tree starting with C_k following the method of the optimum-approaching algorithm and compute M_k. If M_k is calculated for the first time at this stage for this sub)table or if $M_k < M^*$ (where M* designates an M calculated earlier during the same stage for the same sub)table), consider M_k as the new value of M*. Construct a new complete tree by attaching the (sub)tree obtained

to the node to which the considered (sub)table belongs.

1.4 Eliminate from further consideration all trees not yet eliminated, and starting at the node considered by any C_i for which $M^* \leqslant L_i$. If $M^* > L_i$ for one or more $i \neq k$, consider L_p, where p indicates the condition showing the lowest value of $L_i < M^*$ for which M has not yet been computed; consider p as the new value of k and go to 1.3.

1.5 Construct as many new complete trees as (sub)trees have been developed, by replacing in the old complete tree, the old (sub)tree starting at the (sub)table not tagged for optimality by the (sub)trees developed at the present stage for that (sub)table.

2. For each complete tree derived so far and not yet eliminated, consider all nodes not yet tagged for optimality and not belonging to a (sub)tree starting with another node tagged for optimality. Examine for each such node whether $M_k \leqslant L_i$ for the condition C_k tested and all $i \neq k$ at the stage to which the node belongs and at all subsequent stages (cf. Theorem 4). If this condition holds for all $i \neq k$, tag the appropriate node for optimality. (For efficiency reasons, the nodes are best examined from the top to the bottom).

3. For each complete tree derived so far and not yet eliminated, consider the node at stage 1. All trees for which the two nodes at the next stage are tagged for optimality, are considered to belong to set A. If a complete tree is found to belong to set A, then all other complete trees starting with the same condition at node 1 are eliminated from further consideration. From the set B, consisting of set A and a complete tree in storage, store the complete tree showing the lowest average test time and eliminate all other trees

belonging to set B from further consideration. If all complete trees, except the one is storage, are eliminated, the tree in storage is the optimal one and the algorithm is terminated.

4. For each complete tree not yet eliminated and not in storage (let us call the set of all such trees set C), consider all nodes tagged for optimality not belonging to a (sub)tree starting with another node tagged for optimality. For each such node, examine the other complete trees of set C looking for trees showing another condition at that node but the same path from that node to node 1. For all complete trees at set C fulfilling the latter condition, replace the condition at the considered node and the (sub) tree attached to it by the condition at the node tagged for optimality and the (sub)tree attached to it.

5. Consider all trees belonging to set C. If 2 or more trees are found with exactly the same structure, retain only one of them and eliminate the others from consideration.

6. From set C, define a subset D, consisting of all trees from which all nodes belonging to the next stage are tagged for optimality. From the set E consisting of subset D and the tree in storage, store the tree showing the lowest total test time and eliminate all other trees belonging to set E from further consideration.
   From set C, define a subset F, consisting of all trees not belonging to subset D. For all trees of subset F, calculate K, which is the sum of 1) the values T_i for the conditions tested at the present and earlier stages and 2) the values T_i for conditions tested at later stages if these conditions belong to a subtree from which the node at the next stage is tagged for optimality and 3) the values T_i for conditions tested at the next stage but not belonging to a subtree tagged for optimality at the next stage. Eliminate from

further consideration all trees belonging to subset F, for which S + K $\geqslant$ the total test time of the tree in storage.

7. If the set of trees not yet eliminated and not stored is not empty, move to the next stage and go to 1. If however this set is empty, the tree in storage is the optimal one.

As an example, let us apply this algorithm to the table of Figure 1.

Stage 1

Step 1.1

	R_1	R_2	R_3	R_4	R_5			
f_j	10	30	5	20	35	t_i	T_i	L_i
C_1	N	N	-	Y	Y	60	300	362.5
C_2	N	Y	Y*	-	Y*	5	300	430
C_3	N*	-	Y*	N	Y	20	900	900
C_4	N*	N	Y	N*	N	11	330	330

Step 1.2

$L_{i_{min}} = 330.$ Therefore k = 4.

Step 1.3

	R_1	R_2	R_3	R_4	R_5		
f_j	10	30	5	20	35	t_i	T_i
C_1	N	N	-	Y	Y	60	300
C_2	N	Y	Y*	-	Y*	5	300
C_3	N*	-	Y*	N	Y	20	900
C_4	N*	N	Y	N*	N	11	330 ←

N — C_4 — V

	R_1	R_2	R_4	R_5		
f_j	10	30	20	35	t_i	T_i
C_1	N	N	Y	Y	60	0 ←
C_2	N	Y	-	Y*	5	275
C_3	N*	-	N	Y	20	800

	R_3
f_j	5
C_1	-
C_2	Y*
C_3	Y*

N — C_1 — Y

	R_1	R_2		
f_j	10	30	t_i	T_i
C_2	N	Y	5	0 ←
C_3	N*	-	20	800

	R_4	R_5		
f_j	20	35	t_i	T_i
C_2	-	Y*	5	275
C_3	N	Y	20	0 ←

N — C_2 — Y

N — C_3 — Y

	R_1
f_j	10
C_3	N*

R_1

	R_2
f_j	30
C_3	-

R_2

	R_4
f_j	20
C_2	-

R_4

	R_5
f_j	35
C_2	Y*

R_5

$M_4 = 330 + 0 + 0 + 0 = 330$

$M^* = M_4 = 330$

Figure 2.

Step1.4

$330 < 362.5$

$330 < 430$

$330 < 900$

We therefore eliminate all trees starting with C_1, C_2 or C_3.

Step 1.5

Not applicable, since we only have 1 complete tree.

Step 2.

Consider stage 1, node 1

$$M_4 < L_1, L_2, L_3$$

Consider stage 2, node 2

$$M_1 < L_2, L_3$$

Consider stage 3, node 3

$$M_2 < L_3$$

Consider stage 3, node 4

$$M_3 < L_2$$

We therefore tag all nodes for optimality.

Step 3.

Set A consists of 1 tree, viz. the one developed (starting by C_4). Therefore all other trees starting by C_4 are eliminated from further consideration. Set B also consists of the same tree; so this tree is stored. Since all trees except the one in storage are eliminated at this point, we can conclude that the tree in storage (starting by C_4) is the optimal one.

9. Application of the algorithms on tables containing an ELSE-rule.

If the table to be treated contains an ELSE-rule, the same algorithms can be applied. In this case it is possible that at the end of a tree, a table is obtained consisting of one column, and showing condition which are not starred or dashed. In that case, all such remaining conditions are to be tested in sequence, and to all answer-exits for which no rule specified in the table is applicable, the ELSE-rule must be attached.

As a matter of fact, when the table contains an ELSE-rule, the optimum-finding algorithm will not lead to the real optimum in all cases. It will be clear however, that if optimization is really important, the table should be constructed such that it does not contain an ELSE-rule. In our opinion, it therefore does not make sense to apply the optimum-finding algorithm to tables containing an ELSE-rule.

As an illustration, the optimum-approaching algorithm is applied below to a table containing an ELSE-rule (Figure 3).

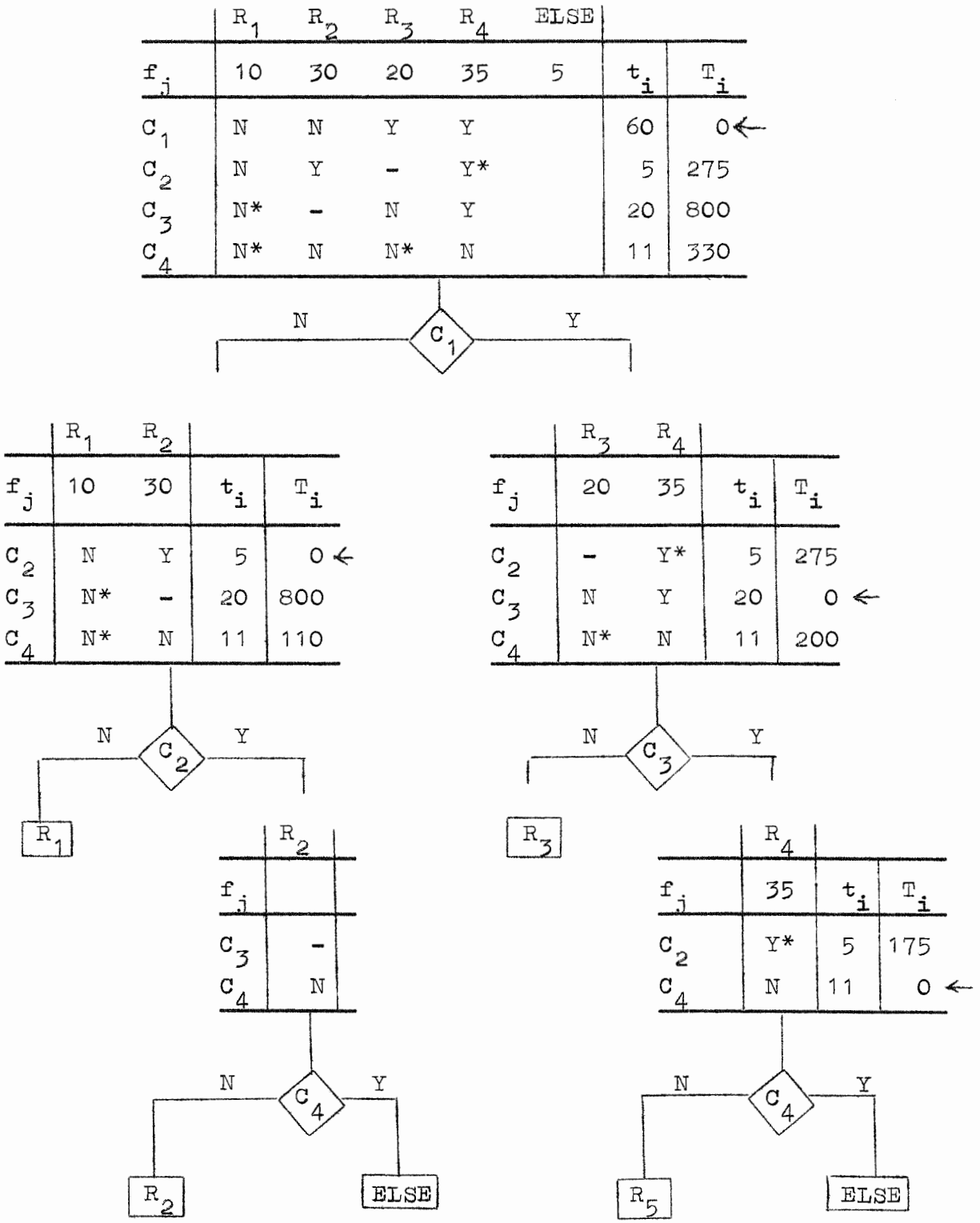

R1 R2 R3 R4 ELSE
fj 10 30 20 35 5 ti Ti
C1 N N Y Y 60 0
C2 N Y - Y* 5 275
C3 N* - N Y 20 800
C4 N* N N* N 11 330
N C1 Y
R1 R2 ti Ti
fj 10 30
C2 N Y 5 0
C3 N* - 20 800
C4 N* N 11 110
N C2 Y
R1
R2
fj
C3 -
C4 N
N C4 Y
R2
ELSE
R3 R4 ti Ti
fj 20 35
C2 - Y* 5 275
C3 N Y 20 0
C4 N* N 11 200
N C3 Y
R3
R4
fj 35 ti Ti
C2 Y* 5 175
C4 N 11 0
N C4 Y
R5
ELSE

Figure 3.

10. Concluding remarks

In order to illustrate our conclusions, we have applied both algorithms proposed in this paper and also the algorithms of Pollack [5] and Shwayder [7] to the following large decision table, derived from a real practical case in the business data processing area.

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}	
f_j	18	10	9	5	10	6	3	1	9	7	10	2	2	8	t_i
C_1	N	N	N*	N*	N*	N	N	N	Y	Y	Y	Y	Y	Y	3
C_2	-	-	N	N	N	Y	Y	Y	Y*	Y*	Y*	Y*	Y*	Y*	8
C_3	N	Y	Y*	Y*	Y*	Y*	Y*	Y*	N	Y	Y	Y*	Y*	Y*	2
C_4	N*	N	Y	Y*	Y*	Y	Y	Y*	N*	N	N	Y	Y	Y*	5
C_5	N*	N*	N	Y	Y	-	N	Y	N*	N*	N*	-	N	Y	10
C_6	-	-	-	N	Y	N	Y	Y	-	N	Y	N	Y	Y	2

The average execution times per transaction are the following.

- testing all conditions for all transactions: 30 (100%)
- algorithm of Pollack: 26.04 (86.68%)
- algorithm of Shwayder: 17.22 (57.40%)
- optimum-approaching algorithm: 16.92 (56.40%)
- optimum-finding algorithm: 16.25 (54.17%)

From this single example alone, no general conclusion can be drawn. This example confirms however what can be derived from a comparison between the two algorithms described in this paper and the other ones mentioned above.

1. The optimum-approaching algorithm will usually lead to a lower execution time than the algorithm proposed by Pollack and Shwayder. Comparing it with the algorithm of Shwayder, it has the advantages to take into consideration the time to

test each condition and to require less computation time. Compared to the algorithm of Pollack, it also shows 2 advantages: it takes into consideration the time to test conditions, and it takes advantage of starred conditions (i.e. conditions on which the answer is known when the answer on one or more other conditions is given.)

2. If the decision table to be converted shows a certain degree of complexity and is somewhat large, the additional computation effort needed to derive the optimal flowchart by applying the optimum-finding algorithm is usually not justified by the small reduction of execution time obtained.

REFERENCES

1. DIXON, Paul. Decision tables and their application. Comput. Automat. 13 (Apr. 1964), 14-19.

2. KING, P.J.H. Conversion of decision tables to computer programs by rule mask techniques. Comm. ACM 9, 11 (Nov. 1966), 796-801.

3. KIRK, H.W. Use of decision tables in computer programming. Comm. ACM 8, 1 (Jan. 1965), 41-43.

4. MUTHUKRISHNAN, C.R. and RAJARAMAN, V. On the conversion of decision tables to computer programs. Comm. ACM 13,6 (June 1970), 247-351.

5. POLLACK, SOLOMON. Conversion of limited-entry decision tables to computer programs. Comm. ACM 8, 11 (Nov. 1965), 677-682.

6. REINWALD, L.T. and SOLAND, R.M. Conversion of limited-entry decision tables to optimal computer programs I: minimum average processing time.J. ACM 13,3 (Jul. 1966), 339-358.

7. SHWAYDER, KEITH.Conversion of limited-entry decision tables to computer programs. A proposed modification to Pollack's algorithm. Comm. ACM 14,2 (Feb.1971), 69-73.

8. VERHELST, M. Procedures for finding optimal and near-optimal test sequences for applying rule mask techniques in object programs derived from decision tables.IAG Quarterly 1, 1 (Jul. 1968), 47-65.